Management of Continuous Spatial Changes

Jose R.R. Viqueira[1], Nikos A. Lorentzos[2], Nieves R. Brisaboa[1]

[1] Computer Science Department, University of A Corunia
Castro de Elvinia S/N, A Corunia, Spain.
Email: {joserios,brisaboa}@udc.es
[2] Informatics Laboratory, Agricultural University of Athens
Iera Odos 75, 11855 Athens, Greece.
Email: lorentzos@aua.gr

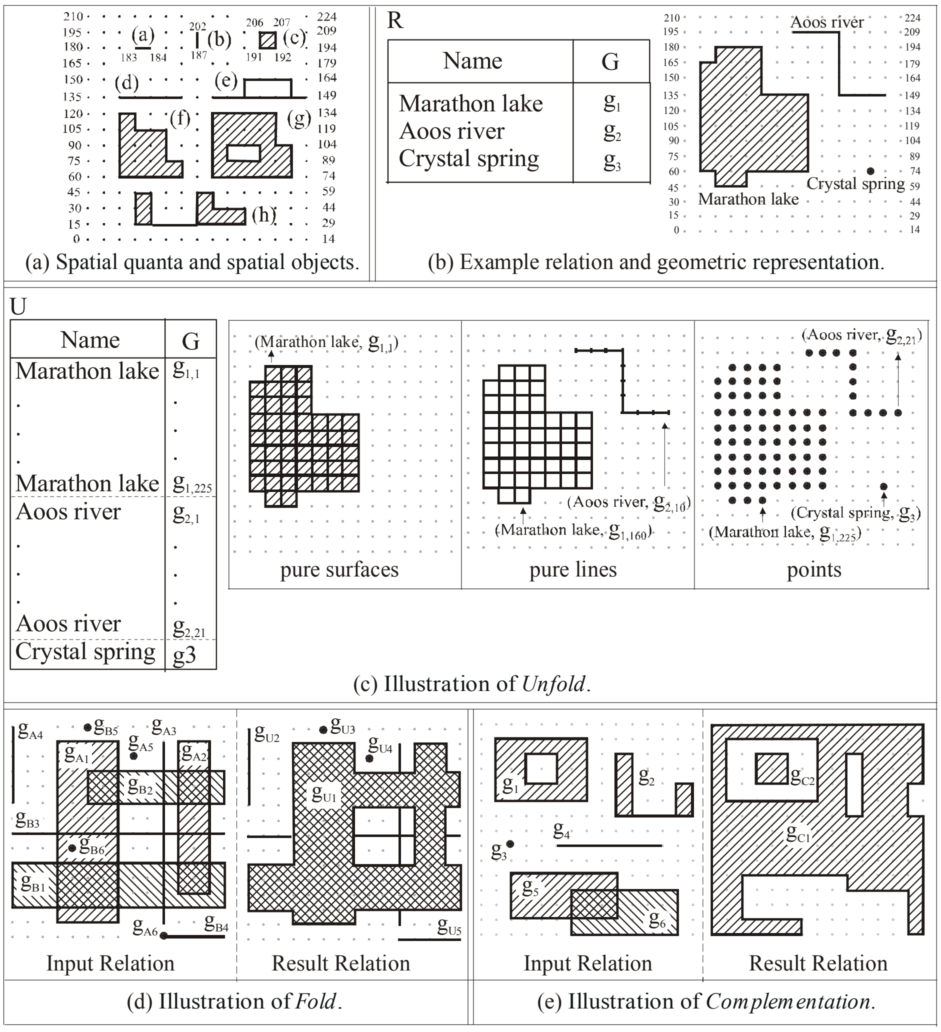
Abstract. Two distinct approaches can be identified in the modelling of spatial data, one that considers *discrete spatial objects* and another that considers *continuous spatial changes*. The first approach fits better cartographic applications whereas the second is commonly used in the modelling of continuous changes of physical phenomena such as temperature, elevation, atmospheric pressure, etc. Due to the fact that the two approaches aim at satisfying distinct user requirements, distinct algebras have also been defined. The present paper aims at bridging the gap between these two approaches. In particular, it considers an SQL extension that has been defined in previous research work for the management of *discrete spatial objects* and shows that it can be applied for the management of *continuous spatial changes*.

1 Introduction

Two distinct approaches can be identified in spatial data modelling, *object-based* and *field-based* [1]. The former view geographic space as being populated by discrete *spatial objects*, e.g. points, lines, surfaces, that have a position in space. Examples of such *spatial objects* are wells, rivers and countries. The latter consider spatial data as mappings from spatial points to domains that represent *spatial properties*. One such domain can be either *discrete* or *continuous*. Examples of discrete domains are soil and vegetation type. Examples of continuous domains are temperature, atmospheric pressure and elevation. Most of research has been dedicated to the definition of models in the first approach but few efforts have been made in the second, from which the most representative piece of work is that by Tomlin [8]. Although the set of operations defined in [8] have not been formalized, they represent a potential standard [6] and serve as a basis for raster based tools [3]. This paper aims at filling in the gap between the two research approaches. In particular, it is shown that an SQL extension defined for the management of interval [4] and spatial data [9, 10] can be applied for the management of *continuous spatial changes*. The remainder of this paper is outlined as follows: In Section 2, *spatial quanta* are defined and, based on them, spatial data types are formalized. Spatial predicates and functions are defined in Section 3. In Section 4, two additional relational algebra operations are defined, *Unfold* and *Fold*. The recursive capabilities of SQL:1999 and part of an SQL extension is outlined in Section 5. In Section 6 it is shown how this extension can be applied to express representative operations of Tomlin's map algebra. Conclusions are drawn in the last section.

2 Spatial Quanta and Spatial Data Types

Consider some n ∈ I, n > 0, with a fixed value. Let $I_n = \{0, 1, \ldots, n-1\}$ and let i, j ∈ I_n. Then there is exactly one integer k, $0 \le k \le n^2-1$, such that $k = n*j + i$. Inversely, it is known that for each such k there is exactly one such pair (i, j) satisfying $k = n*j + i$. Each pair (i, j), equivalently each k, can then be represented in R×R by a dot (Fig. 1(a)). It is said that k is the *ordinal number* of (i, j) and that (i, j) are the *coordinates* of k. Based on this, three types of *spatial quanta* can then be defined.



(a) Spatial quanta and spatial objects.

R

Name	G
Marathon lake	g_1
Aoos river	g_2
Crystal spring	g_3

(b) Example relation and geometric representation.

U

Name	G
Marathon lake	$g_{1,1}$
.	.
.	.
.	.
Marathon lake	$g_{1,225}$
Aoos river	$g_{2,1}$
.	.
.	.
.	.
Aoos river	$g_{2,21}$
Crystal spring	g3

(c) Illustration of *Unfold*.

(d) Illustration of *Fold*.

(e) Illustration of *Complementation*.

Fig. 1. Spatial Quanta, Spatial Data Types and Spatial Relational Formalism.

Quantum Point: Given an integer k, the singleton $P_k = \{k \mid k \in I_{n^2-1}\}$ is called a *2-dimensional* (*2-d*) *spatial point* or a *2-d quantum point* or simply *point*.

The set of all quantum points is denoted as Q_{POINT}. The *coordinates* of P_k are those of k. The geometric representation of P_k is that of its coordinates (i, j). Fig. 1(a) shows the geometric representation of P_k, for k = 0, 1, ..., 224.

Quantum Line: Let $P_k \in Q_{POINT}$ have coordinates (i, j) and let the coordinates of another point be (i+1, j). Then *a pure horizontal quantum line* is defined as the set

$$H_k \equiv \{(x, y) \in R^2 \mid i \leq x \leq i+1 \wedge y = j\}.$$

Similarly, if P_k has coordinates (i, j) and those of another point are (i, j+1) then a *pure vertical quantum line* is defined as the set

$$V_k \equiv \{(x, y) \in R^2 \mid x = i \wedge j \leq y \leq j+1\}.$$

Finally, *pure quantum line* is called any pure horizontal or any pure vertical quantum line. The set of all pure horizontal (pure vertical) quantum lines is denoted as Q_{PH} (Q_{PV}). The set of all pure quantum lines is denoted as Q_{PL}. A pure quantum line can geometrically be interpreted as a line segment. Hence, (a) and (b) in Fig. 1(a) are two pure quantum lines, H_{183}, a pure horizontal quantum line, and V_{187}, a pure vertical quantum line.

Quantum Surface: Let P_k have coordinates (i, j) and let the coordinates of three other points be (i+1, j), (i, j+1), (i+1, j+1). Then a *pure quantum surface* is defined as the set

$$S_k \equiv \{(x, y) \in R^2 \mid i \leq x \leq i + 1 \wedge j \leq y \leq j+1 \}.$$

The set of all pure quantum surfaces is denoted as Q_{PS}. A pure quantum surface can geometrically be interpreted as a square. Hence, (c) in Fig. 1(a) depicts a pure quantum surface, S_{191}.

$Q_{LINE} \equiv Q_{PL} \cup Q_{POINT}$ is called *set of all quantum lines*. $Q_{SURFACE} \equiv Q_{PS} \cup Q_{LINE}$ is called the *set of all quantum surfaces*. An element in $Q_{SURFACE}$ is called *quantum of space* or *spatial quantum*.

Quantum Set: If $\varnothing \neq S = q_1 \cup q_2 \cup \ldots \cup q_n \subset R^2$, where $q_i \in Q_{SURFACE}$ $\forall$ i = 1, 2, …, n, then S is called a *quantum set*.

Spatial Connectivity: A *quantum set* $S \subset R^2$ is *connected* iff for every pair of reals $x, y \in S$ there exists a sequence of quanta $q_1, q_2, ..., q_n \subseteq S$ satisfying the following two properties:

1. $x \in q_1$ and $y \in q_n$.
2. $q_i \cap q_{i+1} \neq \varnothing$ for i = 1, 2, …, n-1.

Spatial Data Types: Let g be a non-empty, connected quantum set. It is then defined that g is of a (*2-d spatial*) type

– POINT	$\Leftrightarrow$	$g \equiv q_i$,	$q_i \in Q_{POINT}$	(e.g. {0}, {2}, ..., {224})
– QPLINE	$\Leftrightarrow$	$g \equiv q_i$,	$q_i \in Q_{PL}$	(e.g. (a), (b))
– QPSURFACE	$\Leftrightarrow$	$g \equiv q_i$,	$q_i \in Q_{PS}$	(e.g. (c))
– PLINE	$\Leftrightarrow$	$g \equiv \cup_i q_i$,	$q_i \in Q_{PL}$	(e.g. (a), (b), (d) (e))
– LINE	$\Leftrightarrow$	$g \equiv \cup_i q_i$,	$q_i \in Q_{LINE}$	(e.g. any pure line and point)
– PSURFACE	$\Leftrightarrow$	$g \equiv \cup_i q_i$,	$q_i \in Q_{PS}$	(e.g. (c), (f), (g))
– SURFACE	$\Leftrightarrow$	$g \equiv \cup_i q_i$,	$q_i \in Q_{SURFACE}$	(e.g. any of the above and (h)).

An element of one of the above types is called, respectively, (*2-d spatial*) *point*, *pure quantum line*, *pure quantum surface*, *pure line*, *line*, *pure surface* and *surface*. The geometric interpretation of the example objects above is depicted in Fig. 1(a).

3 Predicates and Functions

The definitions given below restrict only to those that are used in the remainder sections. If g_1 and g_2 are spatial objects, the following predicates are defined:

- $g_1 = g_2 \Leftrightarrow (\forall\, q \in Q_{SURFACE})((q \subseteq g_1 \Rightarrow q \subseteq g_2) \wedge (q \subseteq g_2 \Rightarrow q \subseteq g_1))$.
- $g_1 <> g_2 \Leftrightarrow \neg(g_1 = g_2)$.
- g_1 *cp* $g_2 \Leftrightarrow g_1 \cap g_2 \neq \varnothing$ (g_1 and g_2 *have common points*).
- g_1 *contains* $g_2 \Leftrightarrow g_1 \subset g_2$.
- *is_point*(g) $\Leftrightarrow g \in$ POINT.

Predicates *is_pure_qline*, *is_pure_qsurface*, *is_pure_line*, *is_line, is_pure_surface* and *is_surface* are defined similarly.

If *q* is a spatial quantum of any type, i.e. either P_k or V_k or H_k or S_k, then the following functions are defined:

- *ord*(q) $\equiv$ k, *p*(k) $\equiv P_k$, *h*(k) $\equiv H_k$, *v*(k) $\equiv V_k$, *s*(k) $\equiv S_k$.
- If *q* is a point P_k (pure quantum surface S_k) then function *north*(P_k) (*north*(S_k)) returns the point (pure quantum surface) closest to the north of P_k (S_k). Functions *south*, *west*, *east*, *northeast*, etc are defined in a similar manner.

If P_{k1}, P_{k2} are points with coordinates (i_{k1}, j_{k1}), (i_{k2}, j_{k2}), the two functions below return, respectively, their Euclidian distance and the slope of the line segment that connects them:

- *distance*$(P_{k1}, P_{k2}) = \sqrt{(i_{k2} - i_{k1})^2 + (j_{k2} - j_{k1})^2}$
- *angle*$(P_{k1}, P_{k2}) =$ *atan*$(i_{k2} - i_{k1}, j_{k2} - j_{k1})$,

where *atan*(x, y) is the function that returns the angle α in the interval [0, 360) whose tangent is defined as *tan*(α) = y/x. If, $P_{B_i} \in$ POINT $\wedge\ P_{A_i} \subseteq g_A \wedge P_{B_i} \subseteq g_B$ then function *distance* is extended so as to apply to any two objects, as follows:

- *distance*$(g_A, g_B) =$ *min*(*distance*(P_{A_i}, P_{B_i}))

4 Relational Formalism

A relation is defined the known way, except that the domain of one or more of its attributes can now be of some spatial data type. G is used to denote an attribute of some spatial data type. **A** denotes a set of attributes of any data type. R(**A**, G) denotes a relation scheme with attributes **A** $\cup$ {G}. Finally, (**a**, g) denotes a tuple of a relation with scheme R(**A**, G). Fig. 1(b) shows the contents of a relation R(Name | CHAR(20), G | SURFACE), as well as the geometric representation of the spatial objects recorded in it. The known set of relational algebra operations is now extended by two more, *Unfold* and *Fold*:

Unfold: It decomposes spatial objects into the set of spatial quanta they consist of. Formally, if R is a relation with scheme R(**A**, G) then

$$U = \textit{Unfold}[G](R)$$

has scheme U(**A**, G), and extension

$$\{(\mathbf{a}, q_i) \mid q_i \in Q_{\mathrm{SURFACE}} \wedge q_i \subseteq g \wedge (\mathbf{a}, g) \in R\}.$$

As an example, if R is the relation in Fig. 1(b), then U = *Unfold*[G](R) yields relation U in Fig. 1(c).

Fold: It *merges* spatial objects. Formally, if R is a relation with scheme R(**A**, G) then relation

$$F = \textit{Fold}[G](R)$$

has scheme F(**A**, G) and extension

$$\{(\mathbf{a},\ g = \textstyle\bigcup_{i=1}^{n} g_i\ ) \mid (g \text{ is } \textit{connected}) \wedge ((\mathbf{a}, g_i) \in R, i = 1, 2, ..., n) \wedge ((\nexists\, (\mathbf{a}, g_k) \in R, k \neq 1, 2, \ldots, n)(g \cup g_k \text{ is } \textit{connected}))\}.$$

As an example, if R(**A**, G) consists of the set of tuples $\{(\mathbf{a}, g_{Ai})\} \cup \{(\mathbf{a}, g_{Bj})\}$, where the spatial objects g_{Ai} and g_{Bj} are those depicted in Fig. 1(d), then F = *Fold*[G](R) consists of the set of tuples $\{(\mathbf{a}, g_{Uk})\}$, where the spatial objects g_{Uk} are also shown in Fig. 1(d).

Many other operations of practical interest for the management of spatial data can be expressed in terms of the previous seven [9]. One of them is operation *Complementation*, whose functionality is illustrated in Fig 1(e). An equivalent SQL:1999 shorthand of this operation in given in Subsection 5.2.

5 Recursion in SQL:1999 and Spatial Extension

In this section it is illustrated how the recursive capabilities of SQL:1999 can be combined with an extension of it, which is based on the algebraic operations defined in the previous section, in a way that enables the management of continuous changes in space.

5.1 Recursive Queries in SQL:1999

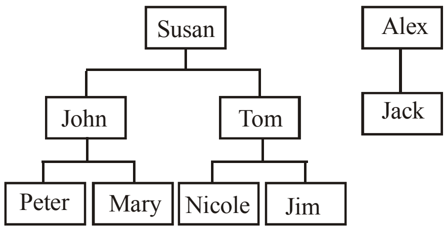
Consider relation EMPLOYEE in Fig. 2 and the following query recursive SQL:1999 query:

```
WITH RECURSIVE DEPENDANT (Emp, Sal, Mgr) AS                    (1)
(                                                              (2)
  SELECT Name AS Emp, Salary AS Sal, Manager AS Mgr            (3)
  FROM EMPLOYEE                                                (4)
  WHERE E.Manager = 'Susan'                                    (5)
 UNION                                                         (6)
  SELECT E.Name AS Emp, E.Salary AS Sal, D.Mgr AS Mgr          (7)
  FROM EMPLOYEE AS E, DEPENDANT AS D                           (8)
  WHERE E.Manager = D.Emp                                      (9)
)                                                             (10)
SELECT sum(Sal)                                               (11)
FROM DEPENDANT                                                (12)
```

EMPLOYEE

Name	Salary	Manager
John	10k	Susan
Susan	20k	
Peter	15k	John
Tom	12k	Susan
Mary	40k	John
Nicole	200k	Tom
Jim	12k	Tom
Jack	190k	Alex
Alex	15k	

(a)



(b)

Fig. 2. Illustration of recursive queries in SQL:1999.

The SQL statement in lines (3)-(9) is executed recursively until the contents of relation DEPENDANT, which is obtained, does not change in two consecutive recursion rounds. It is noticed that during the first execution round, lines (3)-(5) yield the set of tuples

r1: {(John, 10k, Susan), (Tom, 12k, Susan)}

whereas lines (7)-(9) retrieve the empty relation.

In the second round, lines (3)-(5) yield again the tuples of the previous round but now lines (7)-(9) yield the set of tuples {(Peter, 15k, Susan), (Mary, 40k, Susan), (Nicole, 200k, Susan), (Jim, 12k, Susan)}. Therefore, the relation resulting from this round consists of

r2: {(John, 10k, Susan), (Tom, 12k, Susan),
(Peter, 15k, Susan), (Mary, 40k, Susan),
(Nicole, 200k, Susan), (Jim, 12k, Susan)}.

A final third round does not add any new tuples to relation DEPENDANT, hence the recursion phase terminates. Then the statement in lines (11)-(12) is executed, which yields the single tuple {(289k)}, which answers the query "*Give the sum of the salaries of all the employees who are supervised either directly or indirectly by Susan.*"

Given that only *linear recursion* is allowed, a recursive relation (DEPENDANT in this example) may not appear more than once in either the FROM clause or in subqueries of the same query specification. An in depth description of the recursive capabilities of SQL:1999 can be found in [5].

5.2 Spatial Extension

Based on the relational formalism of Section 4, a part of an SQL:1999 extension for spatial data management [9, 10] is now defined, use of which is made in Section 6. This extension supports all the spatial data types, predicates and functions defined in Section 2.

Query Specification: The extended syntax allows the application of operations *Unfold* and *Fold* to the result of a query specification. Thus, if <select> denotes the SQL expression SELECT-FROM-WHERE-GROUP BY-HAVING, and G is a spatial attribute that appears after the keyword SELECT of <select>, then the expressions

<select> **REFORMAT AS UNFOLD** G <select> **REFORMAT AS FOLD** G

apply, respectively, operations *Unfold* and *Fold* to the result obtained by <select>, on attribute G.

Complementation: It enables obtaining the *spatial complementation* of spatial objects. Thus, if <table reference> is a reference to a table with scheme T(**A**, G), and SURF_ALL(G | QPSURFACE) is a relation consisting of all the pure quantum surfaces in $Q_{SURFACE}$, then

<table reference> **COMPLEMENTATION OF** (G)

is defined as a shorthand of the expression

```
SELECT T.A, S.G
FROM <table reference> AS T, SURF_ALL AS S
WHERE S.G NOT IN  (SELECT T1.G
                   FROM <table reference> AS T1
                   WHERE T1.A = T.A
                   REFORMAT AS UNFOLD T1.G)
REFORMAT AS FOLD G
```

As an example, if R(A, G) consists of the set of tuples {(**a**, g_i)}, where the geometric representation of g_i is shown in Fig. 1(e), then the next expression yields the set {(**a**, g_{Ci})}, where each g_{Ci} is also shown in Fig. 1(e).

R **COMPLEMENTATION OF** (G).

6 Application to Continuous Spatial Changes

Now it is shown that the SQL:1999 extension of the previous section can be applied to express the most representative operations of Tomlin's map algebra. In this algebra a *map* is composed of *zones*. Each *zone* is composed of a set of *locations* (elements of the underlying 2-d space) and has an associated numeric value. The algebra consists of about 50 operations, classified into four groups, *Local*, *Zonal*, *Focal* and *Incremental*:

- *Local Operations*: The result value of each *location p* depends on the value of the same *location p* in one or various input *maps*.
- *Zonal Operations*: The result value of each *location p* depends on the values of the *locations* contained in the *zone* of *p* in one or various input *maps*.
- *Focal Operations*: The result value of each *location p* depends on the values of the *locations* contained in the *neighbourhood* of *p* in one or various input *maps*.
- *Incremental Operations*: They extend the set of Focal operations. They take into account the type of *zone* at each *location* (point, line or surface).

Some of the these operations are illustrated in Figs. 3 and 4. With respect to the formalism developed, a *map m* can be seen as a relation R with scheme R(A | REAL, G | QPSURFACE). Each tuple of R records the A value of each *location* (pure quantum surface) in *map m*. A *zone* of R is a set of tuples with the same value for attribute A. A *Digital Elevation Model* (DEM) [1] is a *map* where the value of each *location* represents an elevation above sea level. Although the operations are general, in that they can be applied to any kind of continuous spatial changes, the illustrations provided in this paper restrict to the management of data derived from the DEM of a surface *s*.

6.1 Operation *FocalGravitation*

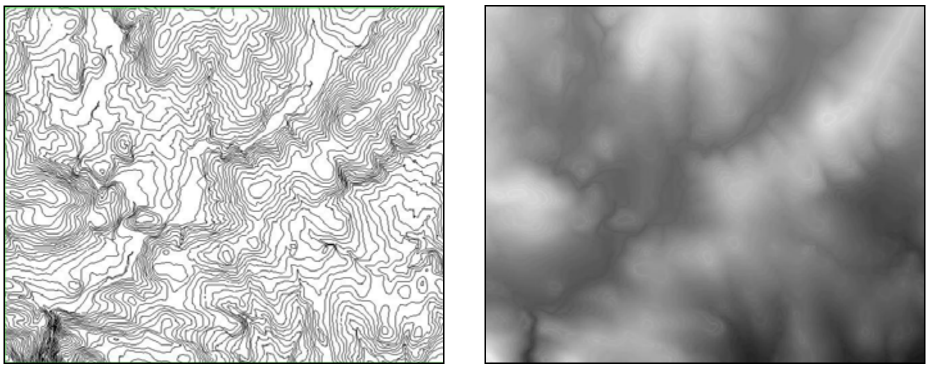
FocalGravitation makes use of the *Inverse Distance Weight* (IDW). It is an interpolation method that obtains a *map m* of continuous changes in space from a sparse set of pairs (a_i, p_i), where p_i is a *location* and a_i is a numeric value. Given a fixed real number d, in the general case, the value a of each *location p* of the output *map m* is given by the formula

$$a = (\sum_i a_i/d_i)/(\sum_i 1/d_i), \text{ where } 0 < d_i = \textit{distance}(p, p_i) < d .$$

Now let R(A | REAL, G | POINT) be a DEM and let d be a real number. Assume also that S(G | QPSURFACE) is a relation consisting of all the pure quantum surfaces of a given rectangular surface *s*, such that all the points in R are contained in *s*. Then the following SQL statement assigns a real value number to each quantum surface in S by the use of the IDW interpolation method.

```
SELECT min(R.A), S.G                                        (1)
 FROM R, S                                                  (2)
 WHERE R.G cp S.G                                           (3)
 GROUP BY S.G                                               (4)
UNION                                                       (5)
 SELECT sum(R.A/distance(R.G, S.G)) /                       (6)
        sum(1/distance(R.G, S.G)) AS A, S.G                 (7)
 FROM R, S                                                  (8)
 WHERE not R.G cp S.G and distance(R.G, S.G) < d            (9)
 GROUP BY S.G                                              (10)
```

In lines (1)-(4) each pure quantum surface in S that contains some point in R.G is assigned the respective R.A value. In case that more than one point is contained in a given pure quantum surface, the minimum A value is assigned. The value of those pure quantum surfaces, which do not contain any point in R, is approximated in lines (6)-(10). For an example, consider relation CONTOUR(A | REAL, G | PLINE) containing the altitude above sea level of each contour line of a given surface *s*. The geometric representation of these lines is depicted in Fig. 3(a). If the above SQL expression is applied to the result of



(a) Contour lines (b) *FocalGravitation*

Fig. 3. Contour lines and DEM obtained with IDW interpolation method.

```
SELECT *
FROM (SELECT *
      FROM CONTOUR
```

```
        REFORMAT AS UNFOLD G)
WHERE is_point(G)
```

with a value of d = 20 then the result is a relation with attributes (A | REAL, G | QPSURFACE) whose contents record the DEM depicted in Fig. 3(b) (darker grey has been chosen to represent lower altitude).

6.2 Operation *FocalMean*

To implement operation *FocalMean*, let DEM(A | REAL, G | QPSURFACE) be a relation recording the DEM whose geometric representation is depicted in Fig. 3(b). Then the following statement retrieves a new DEM, where the value of each pure quantum surface *q* is calculated as the average of the values of all the pure quantum surfaces in DEM whose distance from *q* is less than a given value *d*.

```
SELECT avg(R2.A), R1.G
FROM R AS R1, R AS R2
WHERE distance(R1.G, R2.G) < d
GROUP BY R1.G
```

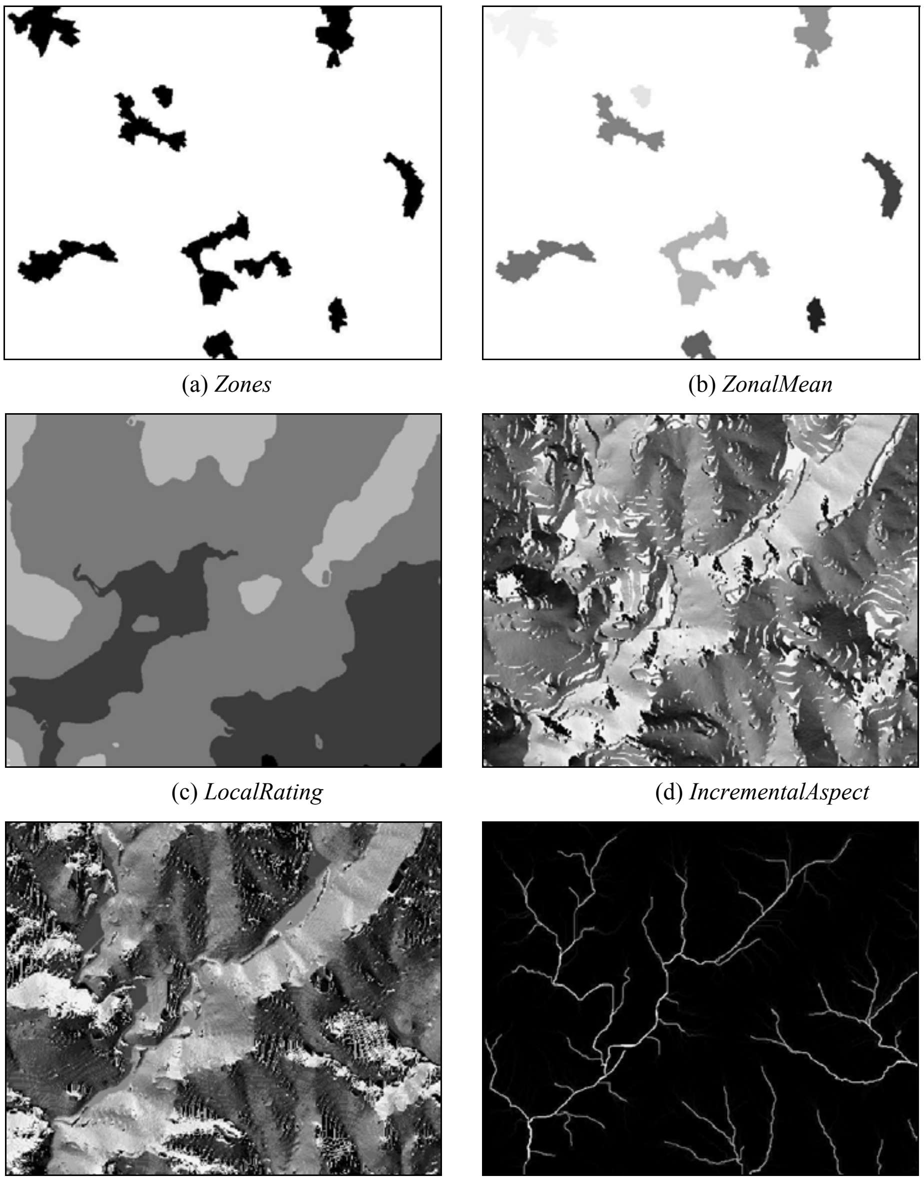
As a result of the application of this statement the DEM in Fig. 3(b) becomes smoother, by the elimination of possible errors produced by the IDW interpolation method. The functionality of other Focal operations such as *FocalMinimum*, *FocalMaximum*, and *FocalSum* can be achieved by just the use of the respective aggregate functions **min**, **max** and **sum**, in place of **avg** of the above SQL statement.

6.3 Operation *ZonalMean*

For operation *ZonalMean*, let R(A | REAL, G | QPSURFACE) be a relation recording the DEM obtained from operation *FocalMean* of the previous subsection. Assume also that PCENTRE(ID | REAL, G | QPSURFACE) is a relation that records a *zone* (set of pure quantum surfaces with the same ID value) for each population centre, i.e. city, town, village, in s. The geometric representation of PCENTRE is depicted in Fig. 4(a). The next statement assigns to each pure quantum surface *q* in each population centre c the average of the altitudes in c.

```
SELECT A, G
FROM (SELECT avg(R.A), P.G
        FROM R,
             (SELECT G
              FROM PCENTRE
              REFORMAT AS FOLD G) AS P
        WHERE P.G contains R.G or P.G = R.G
        GROUP BY P.G
        REFORMAT AS UNFOLD G)
WHERE is_pure_surface(G)
```

The geometric representation of the result of the previous statement is depicted in Fig. 4(b) (darker grey has been chosen to represent lower average altitude). The functionality of other Zonal operations such as *ZonalMinimum*, *ZonalMaximum* and *ZonalSum* can be achieved by just the use of the respective aggregate functions **min**, **max** and **sum** in place of **avg** in the above statement.


(a) *Zones* (b) *ZonalMean*

(c) *LocalRating* (d) *IncrementalAspect*

(e) *IncrementalDrainage* (f) *IncrementalDrainage* with *Spreading* option

Fig. 4. Illustration of examples of *Zonal*, *Local* and *Incremental* operations.

6.4 Operation *LocalRating*

Operation *LocalRating* assigns specific output A values to specific input A values. Considering for example the previous relation R, the following statement assigns an output value of 1 (2, 3, 4, 5) to input values less than 250 (ranging from 250 to 299, ranging from 300 to 349, ranging from 350 to 399, greater than 400).

```
SELECT CASE
        WHEN A<250 THEN                    1
        WHEN A>= 250 and A < 300 THEN 2
        WHEN A>= 300 and A < 350 THEN 3
        WHEN A>= 350 and A < 400 THEN 4
        WHEN A>400 THEN                    5
       END AS A,
       G
FROM R
```

The geometric representation of the result relation is shown in Fig. 4(c) (Darker grey represents lower values of A). Other Local operations such as *LocalSum*, *LocalDifference*, *LocalProduct* etc. can be achieved by applying some arithmetic operations to the respective A values of the same pure quantum surface in various input relations.

6.5 Operation *IncrementalAspect*

Aspect: The *aspect* of a pure quantum surface *q*, in a DEM *m*, is defined as 0, if the *slope* of the surface is 0, and as the direction (angle in the range (0, 360º]) to which the lowest part of *q* is oriented (Fig. 5).

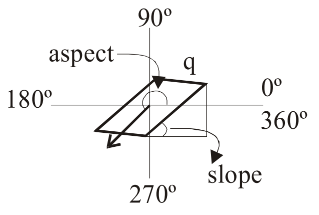


Fig. 5. Illustration of the aspect of a pure quantum surface.

If a_{SW} denotes the elevation of the pure quantum surface *southwest*(q) etc., the *aspect* α of *q* can be approximated by the elevation of the neighbours of *q*, by use of the formula [2]:

$$x = a_{SW} + 2a_W + a_{NW} - a_{SE} - 2a_E - a_{NE}\,,\; y = a_{SW} + 2a_S + a_{SE} - a_{NW} - 2a_N - a_{NE}$$

$$\alpha = \begin{cases} 0 & |\text{ if } x = 0 \text{ and } y = 0 \\ 360 & |\text{ if } atan(x, y) = 0 \\ atan(x, y) & |\text{ otherwise} \end{cases}$$

Now let R be again the relation obtained by operation *FocalMean* in Subsection 6.2. The next statement retrieves the *aspect* of each pure quantum surface in R and matches the functionality of operation *IncrementalAspect*.

```
SELECT CASE                                                      (1)
        WHEN LEFT.A = RIGHT.A and  BOTTOM.A = TOP.A       THEN 0  (2)
        WHEN atan(LEFT.A - RIGHT.A, BOTTOM.A - TOP.A) = 0 THEN 360(3)
        ELSE  atan(LEFT.A - RIGHT.A, BOTTOM.A - TOP.A)            (4)
       END AS A, LEFT.G                                           (5)
FROM (SELECT sum(R1.A *                                          (6)
        (CASE                                                     (7)
          WHEN angle(p(ord(R.G)), p(ord(R1.G))) = 180 THEN 2      (8)
          ELSE 1                                                  (9)
         END)) AS A, R.G                                         (10)
```

```
      FROM R, R AS R1                                             (11)
      WHERE v(ord(R.G)) cp R1.G                                   (12)
      GROUP BY R.A, R.G) AS LEFT,                                 (13)
     (SELECT sum(R1.A *                                           (14)
        (CASE                                                     (15)
          WHEN angle(p(ord(R.G)), p(ord(R1.G))) = 0 THEN 2        (16)
          ELSE 1                                                  (17)
         END)) AS A, R.G                                          (18)
      FROM R, R AS R1                                             (19)
      WHERE v(ord(east(R.G))) cp R1.G                             (20)
      GROUP BY R.A, R.G) AS RIGHT,                                (21)
     (SELECT sum(R1.A *                                           (22)
         (CASE                                                    (23)
           WHEN angle(p(ord(R.G)), p(ord(R1.G))) = 90 THEN 2      (24)
           ELSE 1                                                 (25)
          END)) AS A, R.G                                         (26)
      FROM R, R AS R1                                             (27)
      WHERE h(ord(north(R.G))) cp R1.G                            (28)
      GROUP BY R.A, R.G) AS TOP,                                  (29)
     (SELECT sum(R1.A *                                           (30)
        (CASE                                                     (31)
          WHEN angle(p(ord(R.G)), p(ord(R1.G))) = 270 THEN 2      (32)
          ELSE 1                                                  (33)
         END)) AS A, R.G                                          (34)
      FROM R, R AS R1                                             (35)
      WHERE h(ord(R.G)) cp R1.G                                   (36)
      GROUP BY R.A, R.G) AS BOTTOM                                (37)
WHERE LEFT.G = RIGHT.G and LEFT.G = TOP.G                         (38)
  and LEFT.G = BOTTOM.G                                           (39)
```

For each pure quantum surface *q* in R, relation LEFT in lines (6)-(13) retrieves the value computed by the formula $a_{SW}+2a_W+a_{NW}$. Similarly, RIGHT, TOP and BOTTOM retrieve, respectively, the values of the formulas ($a_{SE}+2a_E+a_{NE}$), ($a_{NW}+2a_N+a_{NE}$) and ($a_{SW}+2a_S+a_{SE}$). The condition in lines (49)-(51) associates the A values retrieved from the above relations to each pure quantum surface. Finally, the expression in lines (1)-(5) retrieves the final value of α for each *q*.

As an example, if R stores the smooth version of the DEM depicted in Fig. 3(b), then the result obtained from the above SQL statement is a relation whose geometric representation is shown in Fig. 4(d). Specifically, the white colour represents value 0 and, for the remainder values, the darker a colour is the less the value it represents.

In a similar manner, operations like *IncrementalSlope* (retrieves the *slope* of each pure quantum surface instead of its *aspect*) can also be expressed in the proposed SQL extension.

6.6 Operation *IncrementalDrainage*

A *depression* in a DEM is a set of *adjacent* pure quantum surfaces, with the same value *a* for attribute A, that are surrounded by pure quantum surfaces whose values for A are greater than *a*. *Depressions* typically define the position of lakes, however, they might also be the result of errors of the interpolation method. A *depression* is filled when its altitude value is raised up to the altitude value of its minimum outlet[1].

[1] Functionality for filling depressions in DEM is typically provided in systems designed for the management of continuous changes in space.

Assuming that a relation R is *depressionless*, the following statement retrieves, for each quantum surface *q* in R, the *flow direction* (either of 0, 45, 90, 135, 180, 225, 270, 315) through which *q* would drain water.

```
WITH RECURSIVE S(N, A, D, G) AS                                      (1)
 (                                                                   (2)
   SELECT 0 AS N, R1.A, angle(p(ord(R1.G)), p(ord(N.G))) AS D, R1.G(3)
   FROM R AS R1, R AS N                                              (4)
   WHERE R1.G cp N.G and R1.G <> N.G and R1.A > N.A                  (5)
     and N.A = (SELECT min(A)                                        (6)
                FROM   R                                             (7)
                WHERE R.G cp R1.G and R.G <> R1.G)                   (8)
     and 1 = (SELECT count(*)                                        (9)
              FROM R                                                (10)
              WHERE R.G cp R1.G and R.G <> R1.G and R.A = N.A)      (11)
  UNION                                                             (12)
   SELECT 0 AS N, R.A, CASE avg(angle(p(ord(R.G)), p(ord(C.G))))    (13)
                         WHEN 120 THEN 0                            (14)
                         WHEN 117 THEN 45                           (15)
                         WHEN 171 THEN 315                          (16)
                         ELSE avg(angle(p(ord(R.G)), p(ord(C.G))))(17)
                       END AS D, R.G                                (18)
   FROM R, (SELECT G                                                (19)
            FROM R COMPLEMENTATION OF (G)                           (20)
            REFORMAT AS UNFOLD G) AS C                              (21)
   WHERE is_pure_surface(C.G) and R.G cp C.G                        (22)
     and R.A <= (SELECT min(A)                                      (23)
                 FROM   R AS TR                                     (24)
                 WHERE R.G cp TR.G and R.G <> TR.G)                 (25)
   GROUP BY R.A, R.G                                                (26)
  UNION                                                             (27)
   SELECT S.N+1 AS N, R.A, angle(p(ord(R.G)),p(ord(S.G))) AS D, R.G  (28)
   FROM R, S                                                        (29)
   WHERE R.G cp S.G and R.G <> S.G and R.A > S.A                    (30)
     and S.A = (SELECT min(A) S.A                                   (31)
                FROM R AS TR                                        (32)
                WHERE TR.G cp R.G and TR.G <> R.G)                  (33)
     and 1 < (SELECT count(*)                                       (34)
              FROM R AS TR                                          (35)
              WHERE TR.G cp R.G and TR.G <> R.G and TR.A = S.A)     (36)
     and angle(p(ord(S.G)), p(ord(R.G))) <> S.D                     (37)
)                                                                   (38)
SELECT min(D), G                                                    (39)
FROM S                                                              (40)
WHERE S.N = (SELECT min(TS.N)                                       (41)
             FROM S AS TS                                           (42)
             WHERE TS.G = S.G)                                      (43)
GROUP BY G                                                          (44)
```

The set of pure quantum surfaces in R are classified in three different groups that are treated with a different method. Each pure quantum surface q_1 in the first group has only one *adjacent* pure quantum surface q_1' whose value for attribute A is the least of all the neighbours of q_1. In this case, q_1 drains to the direction of q_1', that is, to the direction of the steepest drop. A pure quantum surface in this group is treated in lines (3)-(11). The second group concerns some of the pure quantum surfaces in the *boundary* of the rectangular surface *s*. In particular, it concerns those q_2 whose value for A is less than or equal to the least value for A of all its neighbours. In this case, the *flow direction* of q_2 is to the *outside* of surface *s*. The pure quantum surfaces of this group are treated in the lines (13)-(26). Note that the *flow direction* of q_2 is computed in lines (13)-(18) as the average of the various angles that point $p(ord(q_2))$ forms with the points $p(ord(q_{ci}))$, where q_{ci} are the pure quantum surfaces of the *complementation* of *s* which satisfy q_2 *cp* q_{ci}. It has to be noted that this average does not work for three cases that correspond to the pure quantum surface to the northwest corner of *s*, the pure quantum surface to the southeast corner of *s* and to all the pure quantum surfaces to the east side of *s*. The *flow direction* of each of the remainder pure quantum surfaces q_3 is obtained, recursively, in lines (39)-(56). Each q_3 drains to that q_3' of all its neighbours, for which the following conditions hold:

1. q_3' has already a *flow direction*.
2. q_3' does not drain to the direction of q_3.
3. the value for A of q_3' is the least of the values for attribute A of all the neighbours of q_3.
4. If more than one q_3' exists, then q_3 drains to that direction that was calculated in the earliest recursive round. The value of this recursion round is recorded in attribute N of relation S. In case that more than two q_3' have been calculated in the same recursive round, then the least angle is chosen (see lines (39)-(44)).

For an example, assume that relation R records a *depressionless* version of the DEM depicted in Fig. 3(b). Then the result of the above SQL statement is a relation whose geometric representation is shown in Fig. 4(e) (a different degree of grey is assigned to each of the possible directions).

The above functionality resembles the functionality of Tomlin's operation *IncrementalDrainage*. The difference is that more than one *flow directions* can be assigned to a single *location* in *IncrementalDrainage* whereas only one is assigned to each pure quantum surface by the above SQL expression. A detailed description of the problem can be found in [7].

Spreading option. Once the *flow direction* of each pure quantum surface has been approximated, it is possible to obtain, for a given pure quantum surface *q*, the number of pure quantum surfaces in a direct uphill path, starting from *q*. This functionality, commonly called *flow accumulation*, resembles the one achieved by operation *IncrementalDrainage spreading in* and can be obtained by the following recursive SQL statement.

```
WITH RECURSIVE S(C, D, G) AS                                          (1)
 (                                                                    (2)
  SELECT R.D, 1 AS C, R.G                                             (3)
  FROM R                                                              (4)
  WHERE not EXISTS (SELECT *                                          (5)
                    FROM R AS TR                                      (6)
                    WHERE R.G cp TR.G and R.G <> TR.G                 (7)
                      and TR.D = angle(p(ord(TR.G)),p(ord(R.G))))     (8)
 UNION                                                                (9)
  SELECT R.D, sum(S.C) + 1 AS C, R.G                                 (10)
  FROM R, S                                                          (11)
  WHERE R.G cp S.G and R.G <> S.G                                    (12)
    and S.D = angle (p(ord(S.G)), p(ord(R.G)))                       (13)
  GROUP BY R.D, R.G                                                  (14)
 )                                                                   (15)
SELECT C, G                                                          (16)
FROM S                                                               (17)
```

In lines (3)-(8) the pure quantum surfaces which do not receive flow from any neighbour are assigned the value of 1. Then, in the recursive part, the value of each pure quantum surface *q* is calculated as the sum of the values of all the neighbours that drain to the direction of *q*. If R records the result of the SQL statement that implements operation *IncrementalDrainage*, then the above statement retrieves a relation whose geometric representation is shown in Fig. 4(f) (darker grey represents lower value).

7 Conclusions

It has been shown that an SQL extension, originally defined for the management of discrete spatial data, can be applied for the management of *continuous changes in space*. This was illustrated by expressing representative operations for *continuous spatial changes* in this extension. It is estimated that all the operations of Tomlin's map algebra [8] can be expressed in a similar manner. Moreover, it is estimated that additional operations, not included in this algebra, can also be supported. The SQL extension actually enables the uniform management of both interval, temporal, spatial, and spatio-temporal data. At the physical level, vector representations can mainly be used to implement the spatial types POINT, PLINE and PSURFACE of the present paper whereas raster representations fit better the QPSURFACE and QPLINE types. Implementation, in conjunction with optimization techniques, is an issue of further research.

References

[1] P.A. Burrough, R.A. McDonnell, *Principles of Geographical Information Systems*, Spatial Information Systems Series, Oxford University Press, (1998)

[2] B.K.P. Horn, "Hill Shading and the Reflectance Map", *Proceedings of the IEEE* 69(1):14-47, (1981)

[3] MFWorks, URL: *http://www.keigansystems.com/tech.html*, Keigan Systems, (2002)

[4] N.A. Lorentzos, Y.G. Mitsopoulos, "SQL Extension for Interval Data", *IEEE Transactions on Knowledge and Data Engineering* 9(3):480-499, (1997)

[5] J. Melton, A.R. Simon, *SQL:1999 - Understanding Relational Language Components*, Morgan Kaufmann Publishers, Academic Press, (2002)

[6] The OpenGIS Abstract Specification Topic 6: The Coverage Type and its Subtypes, ver.6, OpenGIS Project Document 00-106, Open GIS Consortium Inc., (2000)

[7] D.G. Tarboton, D.P. Ames, "Advances in the Mapping of Flow Networks from Digital Elevation Data," *World Water and Environmental Resources Congress*, (2001)

[8] C.D. Tomlin, *Geographic Information Systems and Cartographic Modeling*, Prentice Hall, (1990)

[9] J.R.R. Viqueira, Formal Extension of the Relational Model for the Management of Spatial and Spatio-temporal Data, Ph.D. Thesis, Computer Science Department, University of A Corunia, (2003)

[10] J.R.R. Viqueira, N.A. Lorentzos, "Spatio-temporal SQL Extension", *Proceedings 8th Panhellenic Conference on Informatics*, Vol.1, Cyprus, November, (2001) 264-273